Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 378 (2007) 237–252
Theoretical Computer Science
www.elsevier.com/locate/tcs

Efficient algorithms for center problems in cactus networks

Boaz Ben-Moshe[a], Binay Bhattacharya[b], Qiaosheng Shi[b,*], Arie Tamir[c]

[a] *School of Computer Science, College of Judea and Samaria, Ariel, 44837, Israel*
[b] *School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, V5A 1S6*
[c] *School of Mathematical Sciences, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel*

Abstract

Efficient algorithms for solving the center problems in weighted cactus networks are presented. In particular, we have proposed the following algorithms for the weighted cactus networks of size n: an $O(n \log n)$ time algorithm to solve the 1-center problem, and an $O(n \log^3 n)$ time algorithm to solve the weighted continuous 2-center problem. We have also provided improved solutions to the general p-center problems in cactus networks. The developed ideas are then applied to solve the obnoxious 1-center problem in weighted cactus networks.

Keywords: Facility location optimization; Center problems; Cactus networks

1. Introduction

In this paper we focus on center location problems in undirected cactus networks. A *cactus* network is a connected graph where any two simple cycles in the graph have at most one vertex in common. In the p-center problem, p centers are to be located in the network so that the maximum weighted distance from a demand point (client) in the network to its nearest center is minimized. To formulate the center location models mathematically, let $X(G)$ represent the candidate location set for the facilities in an undirected network G, and $D(G)$ represent the set of demand points located in G. These locations may occur anywhere along the edges of the network or be restricted to vertices. Each demand point may be weighted by a nonnegative weight. Let $V(G)$ and $A(G)$ denote the set of all vertices and the continuum set of points on the edges of G respectively. A location problem in G will be characterized as a *weighted/unweighted* $X(G)/D(G)/p$ problem where the candidate facility location set and the demand points set are either the set $V(G)$ or the set $A(G)$. Note that the weighted version of a p-center problem is considered only when $D(G) = V(G)$, and therefore weighted $V(G)/A(G)/p$ and weighted $A(G)/A(G)/p$ problems are ignored in this paper.

The p-center problem in general networks is NP-hard [19]. For general networks Kariv and Hakimi [19] proposed $O(mn \log n)$ and $O(mn + n^2 \log n)$ time algorithms for the weighted $A(G)/V(G)/1$ and $V(G)/V(G)/1$ problems

* Corresponding author. Tel.: +1 604 291 5938; fax: +1 604 291 3045.
E-mail addresses: benmo@yosh.ac.il (B. Ben-Moshe), binay@cs.sfu.ca (B. Bhattacharya), qshi1@cs.sfu.ca (Q. Shi), atamir@post.tau.ac.il (A. Tamir).

0304-3975/$ - see front matter
doi:10.1016/j.tcs.2007.02.033

Table 1
Current best results for the p-center problems in tree networks

Problem	Unweighted	Weighted
$V(G)/V(G)/p$	$O(n)$ [11]	$O(n \log^2 n)$ [25] $O(pn \log n)$ [18]
$A(G)/V(G)/p$	$O(n)$ [11]	$O(pn \log n)$ [18] $O(n \log^2 n)$ (combining [24] and [10])
$V(G)/A(G)/p$	$O(n)$ [11]	–
$A(G)/A(G)/p$	$O(pn \log 2n/p)$[12] $O(n \log^2 n)$ (combining [24] and [10])	–

Table 2
Complexity bounds of the algorithms presented in this paper for the p-center problems in cactus networks

Problems	Unweighted	Weighted
$V(G)/V(G)/1$	–	$O(n \log n)$
$V(G)/A(G)/1$	$O(n)$	–
$A(G)/A(G)/1$	$O(n)$	–
$A(G)/V(G)/1$	–	$O(n \log n)$
$A(G)/V(G)/2$	–	$O(n \log^3 n)$
Obnoxious $A(G)/V(G)/1$	–	$O(n \log^3 n)$
$V(G)/V(G)/p$	–	$O(n \log^2 n)$
$V(G)/A(G)/p$	$O(n^2)$	
$A(G)/V(G)/p$	–	$O(n^2)$
$A(G)/A(G)/p$	$O(n^2 \log^2 n)$	–

respectively. They then proposed an $O(m^p n^{2p-1} \log n/(p-1)!)$ algorithm to solve the weighted $A(G)/V(G)/p$ problem in general networks. Later, Tamir [28] improved the above bound to $O(m^p n^p \log n \alpha(n))$ where $\alpha(n)$ is the inverse Ackermann's function.

The location problems in tree networks are well studied [11,12,19,22,24]. Table 1 summarizes the current best results for the p-center problems in trees.

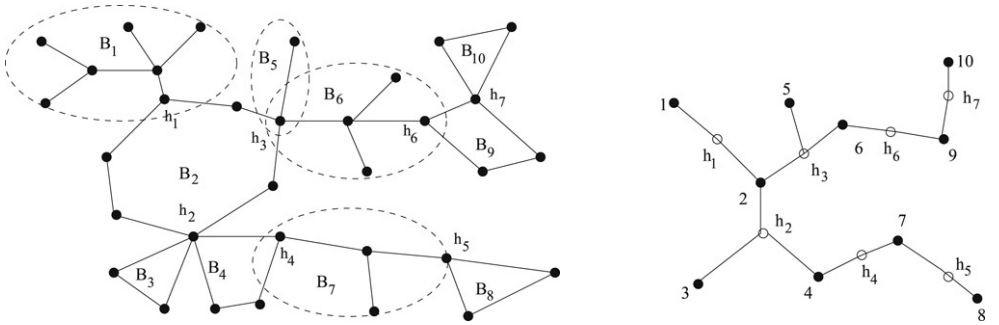
Although most of the reported works on the center problems are for trees or for general networks, more and more attention has been paid to the classes of networks that are between these two extremes [14]. The location problems in cactus networks [12,20,32], in series–parallel networks [16], and in partial k-trees [13] are worth mentioning. When the demand points are unweighted, Lan et al. [20] designed a linear time algorithm to solve the unweighted $V(G)/V(G)/1$ problem in cactus networks [20]. In [3] Burkard and Dollani solved the unweighted $A(G)/V(G)/1$ problem in cactus networks in linear time. Frederickson and Johnson [12] solved the unweighted $V(G)/V(G)/p$ problem in cactus networks in $O(n \log n)$ time.

A center is called *obnoxious* if it maximizes the minimum weighted distance of the demand points to the center. For trees, Tamir [28,29] gives two algorithms of $O(sn \log^2 n)$ and $O(n \log^2 n)$ time, respectively, where s is a parameter that depends on the structure of the tree. The first algorithm was later improved to $O(sn \log n)$ [4]. Zmazek and Žerovnik [32] proposed an algorithm that finds the obnoxious 1-center problem in cactus networks in $O(cn)$ time, where c is the number of distinct vertex weights.

Table 2 summarizes the results reported in this paper.

The basic technique used in developing the algorithms is a combination of a divide-and-conquer technique with parametric searching. One important feature of the algorithm to solve the 2-center problem is to compute, in the query mode, the service cost of an arbitrary point in G in sublinear time. We have proposed a two-level tree decomposition structure on a cactus network for this purpose. This structure can be easily extended to general partial k-tree networks for a fixed k.

The rest of the paper is organized as follows. Section 2 begins with definitions and problem formulations. The well-known tree structure of a cactus graph is also reviewed in this section. Section 3 provides a simple $O(n \log n)$-time



(a) Cactus graph G. (b) Tree structure T_G over G.

Fig. 1. A cactus graph and its corresponding tree structure.

algorithm to solve the weighted $A(G)/V(G)/1$ and $V(G)/V(G)/1$ problems in cactus networks. We then show that the solutions to other 1-center problems mentioned in Table 2 follow quite easily. Our algorithms for the weighted $V(G)/V(G)/2$ and $A(G)/V(G)/2$ problems in cactus networks are presented in Section 4. Section 5 describes some new results on the weighted/unweighted p-center problems in cactus networks. Section 6 gives a brief summary of the results and future possibilities.

2. Definitions and problem formulations

Let $G = (V(G), E(G), w, l)$ be a simple (i.e., no parallel edges and self-loops) cactus network with vertex set $V(G)$, $|V(G)| = n$, and edge set $E(G)$, $|E(G)| = m$, where each vertex $v \in V(G)$ is associated with a nonnegative weight $w(v)$ and each edge $e \in E(G)$ is associated with a positive length $l(e)$ ($|G| = |V(G)|$). We also use $\overline{uv}$ to denote edge e if u and v are the two incident vertices of e. An edge is identified with a line segment of length $l(e = \overline{uv})$ so that any "*point*" on $\overline{uv}$ at a distance t from u and $l(e) - t$ from v can be referred. The set of all such points of the network is denoted by $A(G)$. For a subnetwork G' of G, let $V(G')$, $E(G')$, $A(G')$ denote the vertex set of G', the edge set of G' and the continuum set of points on the edges of G', respectively. For $u, v \in A(G)$, let $P_{u,v}$ denote a *shortest path* in G from u to v; its length is represented by $d_{u,v}$. Let $D(G)$ be the set of the demand points located in G. We consider only two possibilities for $D(G)$: either $D(G) = V(G)$ or $D(G) = A(G)$. As discussed earlier, when $D(G) = A(G)$, we assume that the demand points are unweighted (i.e., weights are the same). We denote the maximum cost of serving the demand points $D(G)$ in G from a point x in G by $r(x, D(G))$, that is,

$$r(x, D(G)) = \max_{v \in D(G)} \{w(v)d_{x,v}\}.$$

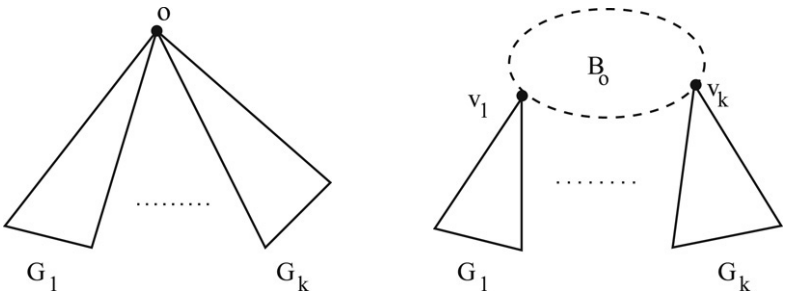
$r(x, D(G))$ is also known as the *radius* of x. The above definition can be generalized to $r(F, D(G))$ where $r(F, D(G))$ denotes the maximum service cost of $D(G)$ from a set of p facilities $F : \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$, i.e.,

$$r(F, D(G)) = \max_{v \in D(G)} \{w(v)d_{F,v}\}, \quad \text{where } d_{F,v} = \min_{j=1,\ldots,p} d_{\alpha_j,v}.$$

In order to facilitate the overview of the proposed algorithms for the center problems in cactus networks, we start with the well-known tree structure of a cactus network [5]. The vertex set $V(G)$ is partitioned into three different subsets. A *C-vertex* is a vertex of degree 2 which is included in exactly one cycle. A *G-vertex* is a vertex not included in any cycle. The remaining vertices, if any, will be referred to as *H-vertices* or *hinges* (See Fig. 1(a).) We use the dotted ellipses to emphasize blocks.

It is not difficult to see that a cactus consists of blocks where each block is either a cycle or a graft (a subtree), and these blocks are glued with H-vertices. So, we can use a tree $T_G = (V_G, E_G)$ to represent the important structure of G, where each node in V_G represents a block or a hinge vertex in G (see Fig. 1(b)). Let B_b denote the block represented by a block node $b \in V_G$. There is an edge between a block node b and a hinge node h if $h \in V(B_b)$. In this case we say that B_b is attached to h.

The weighted $A(G)/V(G)/p$ problem (also known as the weighted continuous p-center problem) in a cactus network G seeks to find a set $F : \{\alpha_1, \ldots, \alpha_p\} \subset A(G)$ that minimizes $r(F, V(G))$. When the centers are restricted



(a) o is a hinge vertex. (b) o is a block node.

Fig. 2. Locate the sub-cactus where the center lies.

to the vertices of G, the problem is known as the weighted $V(G)/V(G)/p$ problem (also known as the weighted discrete p-center problem).

For the case when $p = 1$, let $\alpha^*_G \in A(G)$ be an optimal 1-center of G and let γ_G denote the *radius* $r(\alpha^*_G, D(G))$ of G. Let G' be a subgraph of G. We call $x^* \in A(G')$ *optimal local center* in G' of G if $r(x^*, D(G)) = \min_{x\in A(G')} r(x, D(G))$.

3. Weighted discrete and continuous 1-center problems

We know that the radius function $r(\alpha, D(G))$ is convex along any simple path in G if G is a tree [19]. Unfortunately, this convexity property does not hold in cactus networks (even when it is just a cycle) [19]. We show here that a similar convexity property of a 1-center in a cactus can be established by adding extra points in $A(G)$. Let $G_1, \ldots, G_k$ denote the connected components attached to a hinge vertex h. If the maximum radius of $r(h, D(G_1)), \ldots, r(h, D(G_k))$ is attained at more than one component then clearly h itself is an optimal 1-center. On the other hand, if the maximum is attained in a unique G_i, then G_i must contain an optimal 1-center. This allows one to find in linear time whether a given hinge vertex h is an optimal 1-center, and in the case when h is not an optimal 1-center, determines which component attached to h contains an optimal 1-center. The proposed algorithm has two steps which are described in Sections 3.1 and 3.2. Similar steps were discussed in [7] for general networks. Our version here for the cactus networks is slightly modified. Let B^* be the block containing an optimal 1-center α^*_G.

3.1. Locating a block B^*

Let o be a centroid node of T_G, the tree structure of G described earlier. Thus o is a vertex with the property that each of the subtrees of $T(G)$ rooted at o has no more than half of the graft and block nodes together of T_G. o can be easily identified in linear time [19]. Note that a tree might have at most two centroid vertices. When it has two centroids, the two centroid vertices are connected by an edge [15]. The node o could, therefore, be a hinge vertex or a block (either a cycle or a graft) in G. These cases are separately considered below.

Case 1: o is a hinge vertex (Fig. 2(a)). If there exist subnetworks G_i and G_j attached to o, $i \neq j$, such that $r(o, D(G_i)) = r(o, D(G_j)) \geq r(o, D(G_l))$ for every G_l, $l \neq i, j$, attached to o, then o itself is an optimal 1-center α^*_G. Suppose that the subnetwork G_i with the largest $r(o, D(G_i))$ is unique. In this case an optimal center lies in G_i. Clearly, $r(o, D(G_j))$, for all j, $1 \leq j \leq k$ can be evaluated in linear time.

Case 2: o is a block node (Fig. 2(b)). If there are two subnetworks G_i and G_j attached to B_o, such that $r(v_i, D(G_i)) = r(v_j, D(G_j)) \geq r(v_l, D(G_l))$ for every G_l attached to B_o, $l \neq i, j$, then an optimal center lies in the block B_o. In this case B_o is B^*. Suppose that G_i with the largest value of $r(v_i, D(G_i))$ is unique. Therefore an optimal 1-center either lies in block B_o or in sub-cactus G_i. We compare $r(v_i, D(G_i))$ with $r(v_i, D(G \setminus G_i))$. If $r(v_i, D(G_i)) < r(v_i, D(G \setminus G_i))$, then an optimal center certainly lies in block B_o. Similarly, if $r(v_i, D(G_i)) > r(v_i, D(G \setminus G_i))$, then an optimal center lies in G_i. The remaining case is when $r(v_i, D(G_i)) = r(v_i, D(G \setminus G_i))$. In this case, the hinge vertex v_i itself is an optimal center α^*_G. Clearly all of these steps require linear time to compute.

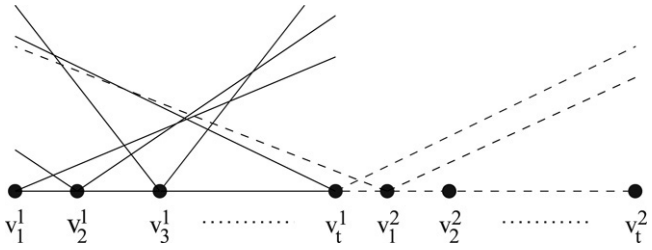


Fig. 3. Locate α^* in a cycle block C.

Thus we have the following situations: either α^*_G is found and in this case the algorithm terminates, and B^* is found; or a sub-cactus G_i containing α^*_G is found and in this case the process is repeated on G_i. The above situation can be tested in linear time. Therefore,

Lemma 3.1. *It takes $O(n \log n)$ time to locate B^*.*

We note that the above process leading to Lemma 3.1 is also valid for general networks. Specifically, we can identify a bi-connected component containing an optimal 1-center of a general network in $O(n \log n)$ time, provided that the distances from a point to all the other demand nodes are computable in linear time. This improves upon the result in [7].

Observation 1. Please note here that the process of identifying B^* can be performed in linear time if the points in $D(G)$ are unweighted. This is due to the fact that, unlike the weighted case, the complement of G_i, i.e. $G \setminus G_i$, can be replaced by just one demand point in $D(G \setminus G_i)$.

3.2. Determining α^*_G in B^*

We now consider the problem of locating α^*_G in B^*. If B^* is a graft, $r(x, D(G))$ is convex on every simple path of B^* [19]. Note that the structure of the cactus network G, except for the part of B^*, can be transformed to an equivalent tree structure. Thus, the $O(n \log n)$-time algorithm in [19] can be used to determine local center α^*_G in B^*. Also the linear-time algorithm for the weighted $V(G)/V(G)/1$ and $A(G)/V(G)/1$ problems in trees [22] can be applied here.

Suppose B^* is a cycle block. Observe that locating α^*_G in the cycle block B^* is very similar to locating the 1-center in a cycle. Rayco et al. [26] in their paper just mentioned that the weighted continuous 1-center problem (i.e. weighted $A(G)/V(G)/1$ problem) in a cycle is solvable in $O(n \log n)$ time. Here, for completeness, we describe an algorithm to solve the weighted 1-center problem in a cycle block.

3.2.1. *Weighted continuous* 1*-center problem in a cycle block*

Let $v_1, v_2, \ldots, v_t$ be the vertices of a cycle C. Let α^* denote an optimal 1-center of C we are interested in computing. We notice that there is exactly one edge in C not used by α^* to cover the vertices of C. We call this edge the *optimal cut-edge* of α^*. Thus the 1-center on the path constructed by removing the optimal cut-edge from C is also an optimal 1-center of C. Thus our idea is to consider each edge as a cut-edge and compute the 1-center on the resulting path. The data structure described below is dynamic, and allows efficient updating of the structure as the cut-edge changes.

The algorithm is described as follows. Consider Fig. 3 for reference. The vertices on the cycle are indexed as $v_1, v_2, \ldots, v_t$ in counterclockwise order and the edge connecting the vertices v_{i-1} and v_i is indexed as e_{i-1}, $1 < i \leq t$ ($e_t = \overline{v_t v_1}$). We put the $2t-1$ vertices $\{v_1^1 = v_1, v_2^1 = v_2, \ldots, v_t^1 = v_t, v_1^2 = v_1, \ldots, v_{t-1}^2 = v_{t-1}\}$ on a real line. Let $\mathrm{Pos}(z)$ denote the position of z on the real line. The positions of the $2t-1$ vertices are determined in the following way. $\mathrm{Pos}(v_1^1) = 0$, $\mathrm{Pos}(v_i^1) = \mathrm{Pos}(v_{i-1}^1) + l(e_{i-1})$, $1 < i \leq t$; and $\mathrm{Pos}(v_1^2) = \mathrm{Pos}(v_t^1) + l(e_t)$, $\mathrm{Pos}(v_i^2) = \mathrm{Pos}(v_{i-1}^2) + l(e_{i-1})$, $1 < i \leq t-1$.

The path constructed by removing edge e_i from C is called the i-th path, which is the path from v_{i+1}^1 to v_i^2. Let V^i be the vertex set of the i-th path. The service cost function $f^i(x)$ on the i-th path is defined as

$$f^i(x) = \max_{v \in V^i} w(v) |d_{x,v_1^1} - \mathrm{Pos}(v)|,$$

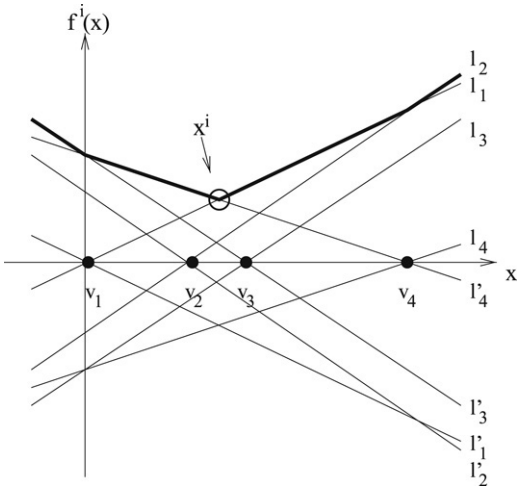


Fig. 4. $f^i(x)$.

where x is a point on the i-th path at a distance d_{x,v_1^1} from v_1^1. Let x^i denote an optimal 1-center of the i-th path. It is easy to compute $f^i(x)$ and determine x^i in linear time [3]. But, it is not efficient to separately compute functions $f^i(x)$, $1 \le i \le t$.

We can represent the function $w(v)|d_{x,v_1^1} - \mathrm{Pos}(v)|$ for all x by two straight lines through the point $(\mathrm{Pos}(v), 0)$ with slopes $w(v)$ and $-w(v)$ (Fig. 4). Then $f^i(x)$ is the upper envelope of $2t$ linear functions where t linear functions have positive slopes and t linear functions have negative slopes (see Fig. 4). Since $f^i(x)$ is convex, the optimal solution can be easily computed. Observe that the upper envelope of the lines generated by the $(i+1)$-th path is constructed from the upper envelope of the lines generated by the i-path by simply removing the lines generated by v_{i+1}^1 and inserting the lines generated by v_{i+1}^2. The upper envelope can be maintained by the algorithm proposed by Hershberger and Suri [17]. Each updating step can be performed in amortized logarithmic time since the sequence of insertions and deletions of lines are already known [17]. Observe that the above approach also works if some of the vertices in C are hinge vertices and are attached to other components. If v is a vertex in a component attached to a hinge vertex, say v_i, the corresponding two lines generated by v will have slopes $w(v)$ and $-w(v)$ and they will go through the point $(\mathrm{Pos}(v_i), b_v)$, where b_v is the weighted distance of v to v_i. Thus

Lemma 3.2. *Optimal solutions corresponding to all the cut-edges in C can be computed in total $O(n \log n)$ time. The storage space requirement is linear.*

Summing up, we have the following theorem.

Theorem 3.3. *The weighted $A(G)/V(G)/1$ and $V(G)/V(G)/1$ problems in cactus networks can be solved in $O(n \log n)$ time using linear space.*

We also have the following result.

Theorem 3.4. *The four unweighted models ($V(G)/V(G)/1$, $V(G)/A(G)/1$, $A(G)/V(G)/1$, $A(G)/A(G)/1$) in cactus networks can be solved in $O(n)$ time using linear space.*

Proof. The result for the models where $D(G) = V(G)$ is in [3,20]. From Observation 1, we note that in $O(n)$ time we can restrict the problems $A(G)/A(G)/1$ and $V(G)/A(G)/1$ to a cactus having at most one cycle. But then in this case the $A(G)/A(G)/1$ problem is equivalent to $A(G)/V(G)/1$ and $V(G)/A(G)/1$ is equivalent to $V(G)/V(G)/1$. □

4. Weighted continuous 2-center problem

In this section, an efficient algorithm for the weighted $A(G)/V(G)/2$ problem in cactus networks is proposed. Let $F = \{\alpha_1, \alpha_2\} \subset A(G)$ be a set of any two centers in G. Let $V_i \subseteq V$ be the set of vertices closest to $\alpha_i \in F$, $i = 1, 2$; ties are broken in such a way that $G(V_i)$ remains connected. The vertices of V_i are thus *covered* or *served* by α_i,

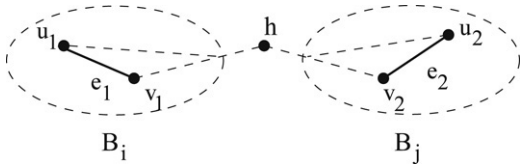


Fig. 5. Lemma 4.1.

$i = 1, 2$. The edges whose endpoints belong to different subgraphs $G(V_i), i = 1, 2$ are called *split-edges*. Thus, locating an optimal 2-center in G is equivalent to finding a set of split-edges whose removal defines two connected components and optimal 1-centers of the resulting two components constitute an optimal 2-center solution of G. The split-edges in an optimal solution are called optimal split-edges.

In a tree network, the number of optimal split-edges is just one. However, for a cactus network the number of optimal split-edges is at most two. As a matter of fact, it can be shown that

Lemma 4.1. *Optimal split-edges in a cactus network G lie in one block B_i. If B_i is graft, there is one optimal split-edge, otherwise (B_i is a cycle) the number of optimal split-edges is two.*

Proof. Suppose that optimal split-edges lie in more than one block. Let α_1 and α_2 be the centers of the subnetworks obtained after the removal of the optimal split-edges from the cactus network. Let B_i and B_j be the blocks containing the split-edges $e_1 = \overline{u_1 v_1}$ and $e_2 = \overline{u_2 v_2}$ respectively (Fig. 5). Assume that u_1 and u_2 are served by α_1, and v_1 and v_2 are served by α_2. Let h be a hinge vertex lying between B_i and B_j, that is, the shortest path between any vertex in B_i and any vertex in B_j passes through h. Such a hinge vertex h always exists since B_i and B_j are two different blocks. Since the subnetwork served by each 1-center is connected, and since h lies in the shortest paths P_{u_1,u_2} and P_{v_1,v_2}, h is served by both α_1 and α_2. This is not possible. Hence, optimal split-edges must lie in one block of G. Therefore, if the block containing the optimal split-edge set is a graft, then there is only one split-edge in the set and if the block containing the optimal split-edge set is a cycle, then there are two split-edges in the set. □

Let R denote a set of split-edges of G where, if $|R| = 2$, both the split-edges come from one cycle block. Let G_R^1, G_R^2 denote the two subnetworks obtained after the split-edges in R are removed from G. Let $\gamma_{G_R^i}$ be the optimal service cost of the $A(G_R^i)/V(G_R^i)/1$ problem, $i = 1, 2$. Let $\phi(R) = \max\{\gamma_{G_R^1}, \gamma_{G_R^2}\}$. A split-edge set R^* is called *an optimal split-edge set* of G if $\phi(R^*) = \min_{R \subseteq B_i, i=1,\ldots,t'} \phi(R)$. Here $B_1, B_2, \ldots, B_{t'}$ are the blocks in G.

4.1. Locating the optimal split-block B^*

We now focus on exploring the properties of the optimal split-edge set in cactus networks.

Lemma 4.2 (*Fig. 2(a)*). *Let $G_1, \ldots, G_k$ be the subnetworks of G attached to a hinge vertex o. In $O(n \log n)$ time we can either identify an optimal split-edge set or determine the subnetwork attached to o that contains an optimal split-edge set.*

Proof. Suppose that $r(o, G_1) \geq r(o, G_2) \geq r(o, G_j)$, $j = 3, \ldots, k$, where $r(o, G_i)$ denotes the service cost of G_i from o. Let R_i denote the set of edges of G_i incident to o, $i = 1, \ldots, k$. Clearly, $|R_i| \leq 2$ for all i. The service cost $\phi(R_j)$ with a split-edge set R_j, $j \neq 1, 2$, must be greater than $\max\{\gamma_{G_1}, r(o, G_2)\}$ since G_1 and G_2, in this case, must be served by the same 1-center. But, the service cost $\phi(R_1)$ is no more than $\max\{r(o, G_2), \gamma_{G_1}\}$. Therefore, there exists an optimal split-edge set in $G_1 \cup G_2$.

In the following we determine whether G_1 or G_2 contains an optimal split-edge set. We consider three cases based on the service costs $\phi(R_1)$ and $\phi(R_2)$.

- $\phi(R_1)$ is determined by the service cost of the center in subnetwork G_1. This implies that G_1 contains an optimal split-edge set.
- $\phi(R_2)$ is determined by the service cost of the center in subnetwork G_2. This implies that G_2 contains an optimal split-edge set.

- $\phi(R_1)$ is determined by the service cost of the center in subnetwork $G \setminus G_1$ and $\phi(R_2)$ is determined by the service cost of the center in subnetwork $G \setminus G_2$. This implies that, if $\phi(R_1) \leq \phi(R_2)$, R_1 is an optimal split-edge set; otherwise ($\phi(R_2) \leq \phi(R_1)$) R_2 is an optimal split-edge set.

In Theorem 3.3 we have shown that the weighted $A(G)/V(G)/1$ problem in cactus networks can be solved in $O(n \log n)$ time. Therefore, it takes $O(n \log n)$ time to either identify an optimal split-edge set or determine the subnetwork that contains an optimal split-edge set. □

Using Lemma 4.2 and a centroid decomposition of T_G [25], we can recursively search the split-block that contains an optimal split-edge set. Thus in $O(n \log n \cdot \log |V_G|)$ time we either identify an optimal split-edge set or determine the block B^* that contains an optimal split-edge set R^*.

4.2. Computing R^* in B^*

When B^* is a graft, R^* contains exactly one edge. We can locate R^* in B^* recursively using a centroid decomposition of B^*. Each recursive step takes $O(n \log n)$ time (Lemma 4.2 also works for G-vertices). Therefore, R^* in a graft B^* can be computed in $O(n \log n \cdot \log |B^*|)$ time.

When B^* is a cycle, an optimal split-edge set contains two edges. Let $v_1, v_2, \ldots, v_t$ be the vertices of B^* in counterclockwise order, and let $e_1 = \overline{v_1 v_2}, \ldots, e_t = \overline{v_t v_1}$ be the edges of B^* in counterclockwise order. Let $[e_i..e_j]$ denote the chain of B^* in counterclockwise order from e_i to e_j in B^*. Similarly, let $[v_i..v_j]$ denote the sequence of vertices in counterclockwise order from v_i to v_j in B^*.

If one of the two optimal split-edges in B^* is known, we can locate the other one in $O(n \log n \cdot \log |B^*|)$ time since it is equivalent to locating one optimal split-edge in a graft. Here the graft is a path. An edge $e'_i \in E(B^*)$ is called a *match-edge* of e_i if $\phi(\{e_i, e'_i\}) = \min_{e_k \in E(B^*)} \phi(\{e_i, e_k\})$. The match-edge of an edge e_i may not be unique, but all the match-edges must be consecutive along the path $\pi : \langle v_{i+1}, v_{i+2}, \ldots, v_1, \ldots, v_{i-1} \rangle$. This is due to the unimodality property of $\phi(\{e_i, e\})$ as e moves away from e_i along the path π. For uniqueness, the last match-edge is paired with e_i. We cannot afford to separately find the match-edge of each edge in B^*. However, the following simple observation is helpful. Assume that e'_i is the match-edge of e_i, $i = 1, \ldots, t$, and $e_j \in [e_i..e'_i]$. The match-edge e'_j of e_j must lie in $[e'_i..e_i]$. This also follows from the unimodality property.

The algorithm to locate R^* in B^* proceeds as follows. The process starts from e_1. After the match-edge e'_i of e_i is found, the first edge $e_j \in [e'_i..e_i]$ that satisfies $\phi(\{e_{i+1}, e_j\}) < \phi(\{e_{i+1}, e_{j+1}\})$ is taken to be the match-edge e'_{i+1} of e_{i+1}. This again follows from the unimodality property. Thus, the running time to compute R^* in B^* is $O(|B^*|)$ times the complexity of computing the maximum service cost $\phi(\{e_i, e_j\})$ for a given split-edge set $R = \{e_i, e_j\}$.

4.2.1. Computing $\phi(R = \{e_i, e_j\})$

Let $v_1, v_2, \ldots, v_t$ be the vertices of B^* in counterclockwise order. Let G'_k denote the subnetwork of B^*, attached to v_k. Assume that $r(v_{i_1}, D(G'_{i_1})) \geq r(v_{i_2}, D(G'_{i_2})) \geq r(v_k, D(G'_k))$, $1 \leq k \leq t$, $i_1 \neq i_2$, and $k \neq i_1, i_2$. The following lemma is crucial to the algorithm of computing $\phi(R = \{e_i, e_j\})$.

Lemma 4.3. *Two centers corresponding to a given split-edge set $R \in B^*$ lie in either block B^*, subnetwork G'_{i_1} or subnetwork G'_{i_2}.*

Proof. Suppose that the vertices $v_{i+1}, \ldots, v_j$ are contained in $V(G^1_R)$. It is clear that an optimal 1-center of G^1_R (resp. G^2_R) lies either in B^* or in some subnetwork G'_k where $k \in [i+1, j]$ and $r(v_k, D(G'_k)) \geq r(v_f, D(G'_f))$, for all $f \in [i+1, j]$ (resp. $k \in [j+1, i]$ and $r(v_k, D(G'_k)) \geq r(v_f, D(G'_f))$, for all $f \in [j+1, i]$).

If possible, suppose that one of the two centers, say α_1, lies in G'_k, where $k \neq i_1, i_2$. Therefore the service cost $\gamma_{G^1_R}$ must be less than or equal to $r(v_k, D(G'_k))$. α_1 cannot serve the vertices v_{i_1} and v_{i_2} since in this case $\gamma_{G^1_R}$ is at least $r(v_{i_2}, D(G'_{i_2}))$, which is greater than $\gamma_{G^1_R}$. Therefore, the service cost $\gamma_{G^2_R}$ must be at least $r(v_{i_2}, D(G'_{i_2}))$. Thus $\gamma_{G^1_R} \leq r(v_k, D(G'_k)) \leq r(v_{i_2}, D(G'_{i_2})) \leq \gamma_{G^2_R}$. Therefore, $r(\{v_k, \alpha_2\}, D(G)) \leq \max\{r(v_k, D(G^1_R)), r(\alpha_2, D(G^2_R))\} = r(\alpha_2, D(G^2_R)) = \phi(R)$. Therefore, we can use the vertex v_k as the center, instead of α_1, without increasing the service cost $\phi(R)$. Hence G'_k where $k \neq i_1, i_2$ can be eliminated from further search. □

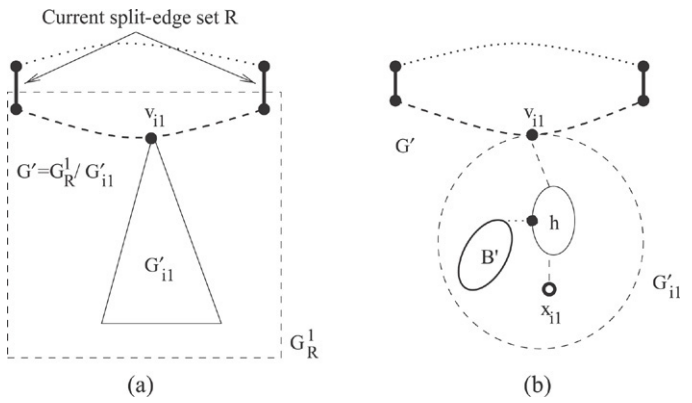


Fig. 6. Example with a split-edge set R.

Suppose that $v_{i_1} \in V(G_R^1)$. We can determine $\phi(R)$ by computing

- an optimal center α_1' of G_R^1 constrained to lie on B^*,
- an optimal center α_1'' of G_R^1 constrained to lie on G'_{i_1},
- an optimal center α_2' of G_R^2 constrained to lie on B^*, and
- an optimal center α_2'' of G_R^2 constrained to lie on G'_{i_2} if $v_{i_2} \in V(G_R^2)$; otherwise α_2'' is undefined.

All α_1' and all α_2' are restricted to be on a cycle block. Hence, they can be found by the algorithm described in Section 3.2.1 in $O(n \log n)$ time.

Since computing α_2'' is similar to computing α_1'', we concentrate on computing α_1'' only. Let $G' = G_R^1 \setminus G'_{i_1}$. G' changes as R changes; see Fig. 6(a). Let x_{i_1} be an optimal 1-center of G'_{i_1}, and B denote the block in G'_{i_1} where x_{i_1} lies.

Lemma 4.4 (*Fig. 6(b)*). *α_1'' lies in one of the blocks that the shortest path $P_{v_{i_1},x_{i_1}}$ goes through.*

Proof. Suppose that α_1'' lies in some block B' that $P_{v_{i_1},x_{i_1}}$ does not go through. Let h denote the closest vertex of the path $P_{v_{i_1},x_{i_1}}$ to α_1''. Clearly, h is a hinge vertex. We can see that the service cost $r(h, D(G_R^1))$ is less than the service cost $r(\alpha_1'', D(G_R^1))$. Therefore α_1'' cannot lie in B'. □

4.2.2. Forcing the convexity of $r(x, D(G'_{i_1}))$ on an edge

Unlike in tree structures, the service cost function $r(x, D(G))$ in a cactus network may not be convex as x moves from one endpoint of the edge to the other [19]. Fortunately, for a cactus network it is possible to force the service cost function to be convex on each edge of the *block path* $P(v_{i_1}, x_{i_1})$, which is a list of blocks that the path $P_{v_{i_1},x_{i_1}}$ goes through. This is achieved by adding extra vertices as follows. If a block on the block path is a graft, then clearly the service cost function is convex on each edge of this block. When a block on the block path is a cycle, for every vertex v in this block, we find its *match-point* v' in the same block such that $d^c_{v,v'} = d^{cc}_{v,v'}$, where $d^c_{v,v'}$ and $d^{cc}_{v,v'}$ are the respective clockwise and counterclockwise distances from v to v' in the block. v' is then added as a vertex to the network by breaking the edge containing v'. We then assign weight zero to these added vertices. In this way, the service cost function $r(x, v)$, for every v, is monotone as x ranges over an edge in the updated network. Due to the introduction of match-points, the service cost function on each edge is therefore convex. The total number of match-points added to force the convexity is no more than $2n$. These match-points can easily be determined in $O(n)$ time.

4.2.3. An algorithm to locate α_1'' in G'_{i_1}

In the following we assume that G'_{i_1} contains the *match-points* vertices in the cycle blocks of the block path. Also, G_R^1, G' and G'_{i_1} are preprocessed along the lines that are discussed in Appendix A. A two-level tree decomposition

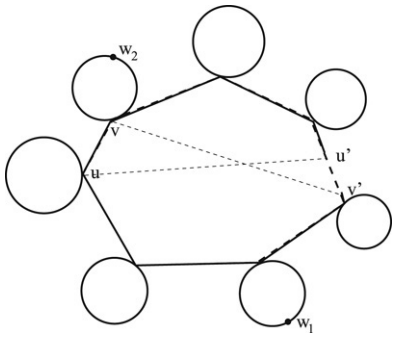


Fig. 7. Local center on an edge $\overline{uv}$.

structure $\overline{TD}(G'_{i_1})$ of G'_{i_1} is built in $O(|G'_{i_1}|\log^2|G'_{i_1}|)$ time. The storage space requirement is $O(|G'_{i_1}|\log|G'_{i_1}|)$. Using this data structure, the service cost of any point in x in G'_{i_1} to $D(G'_{i_1})$ can be answered in $O(\log^2|G'_{i_1}|))$ time.

We first show that the optimal local 1-center of G'_{i_1} on an edge $e = \overline{uv}$ of G'_{i_1} can be computed in $O(\log^3 n)$ time. Let u be the counterclockwise neighbor of v (Fig. 7). Let V_u be the set of vertices in G'_{i_1} which are closer to u than v. Let $V_v = V(G'_{i_1}) \setminus V_u$. The vertices in V_v are closer to v than u. As observed in Appendix A, there are $O(\log n)$ subtrees of $\overline{TD}(G'_{i_1})$, say H, spanning all the vertices of G'_{i_1} and there is a 2-separator (or 1-separator, but it is safe to only consider a 2-separator) between the edge $\overline{uv}$ and each of the subtrees in H. This is possible when we start from a node of $\overline{TD}(G'_{i_1})$ that contains both the vertices u and v. Let w_1 and w_2 be the 2-separator between u and v, and a subtree G'' (an element of H). We can compute d_{u,w_i} and d_{v,w_i} in constant time using the results in [6]. Clearly, if $w_1, w_2 \in V_u$ (or $w_1, w_2 \in V_v$) then all the vertices in G'' belong to V_u (or V_v); if $w_1 \in V_u$, $w_2 \in V_v$ (resp. $w_1 \in V_v$, $w_2 \in V_u$) then all the vertices in G'', whose shortest path to u goes through w_1, belong to V_u (resp. V_v) and the remaining vertices in G'' belong to V_v (resp. V_u). This can be observed in Fig. 7. Let u' and v' be the match-points of u and v, respectively, on the cycle block that contains $\overline{uv}$. All the vertices on the counterclockwise path from u to v' together with the vertices in the components attached to the path are closer to u than v. The shortest paths from u to these vertices do not use the edge $\overline{v'u'}$. These vertices determine V_u. Similarly, all the vertices on the clockwise path from v to u' together with the vertices in the components attached to the path are closer to v than u. The shortest paths from v to all these vertices also do not use the edge $\overline{v'u'}$. These vertices determine V_v. From $\overline{TD}(G'_{i_1})$ the vertices in G'' that belong to V_u can be reported in a sorted list of distances from w_1 in $O(\log n)$ time. Similarly, all distances from w_2 to the vertices in G'' that belong to V_v can be reported in a sorted list in $O(\log n)$ time. Therefore, V_u and V_v can be represented by $O(\log n)$ sorted lists and the maximum service cost function of each such list is monotone on $\overline{uv}$. More precisely, the maximum service cost function of each list in V_u monotonically increases on $\overline{uv}$ from u to v and the maximum service cost function of each vertex in V_v monotonically decreases on $\overline{uv}$ from u to v. Since the maximum service cost to G'_{i_1} of a point on $\overline{uv}$ can be computed in $O(\log^2 n)$ time (Lemma A.1.2),

Lemma 4.5. *The optimal local center of G'_{i_1} constrained to lie on an edge can be computed in $O(\log^3 n)$ time. The preprocessing step takes $O(n\log^2 n)$ time and $O(n\log n)$ space.*

Thus in the following it is assumed that the optimal local center of G'_{i_1} on every edge of G'_{i_1} is already known. The remaining step to compute α''_1 has two parts. We first determine the block that contains α''_1 and then determine α''_1 within this block. Suppose that $u_1 = v_{i_1}, u_2, \ldots, u_k$ are the hinge vertices lying on the path $P_{v_{i_1},x_{i_1}}$.

Locating the block $B_{u_i^*}$ containing α''_1.

Observe that the farthest (weighted) vertex v'_j in G'_{i_1} to u_j must lie below u_j (further away from v_{i_1} compared to u_j); otherwise, x_{i_1} cannot be a weighted 1-center of G'_{i_1}. Therefore, we can conclude that

Lemma 4.6. *For any j, $1 \le j \le k$, if the farthest (weighted) vertex to u_j in G^1_R comes from $G' = G^1_R \setminus G'_{i_1}$, then α''_1 cannot lie on block path $P(u_j, x_{i_1})$; otherwise, i.e., the farthest (weighted) vertex to u_j lies in G'_{i_1}, α''_1 cannot lie on the block path $P(v_{i_1}, u_j)$.*

Using Lemma 4.6, it is easy to locate the block that contains α_1'' in $O(\log n)$ steps. In each step, we need to compute $r(u_i, D(G_R^1))$. $r(u_i, D(G'_{i_1}))$ for any u_i can be computed in $O(\log^2 n)$ time using the structure $\overline{TD}(G'_{i_1})$. Since $r(u_i, D(G_R^1)) = \max\{r(u_i, D(G')), r(u_i, D(G'_{i_1}))\}$, we need to compute $r(u_i, G')$ fast. Observe that the upper envelope $\overline{f^i}(x)$ of the lines generated by the vertices of G' can be dynamically maintained. There are $O(n)$ insertions and deletions in all, and each operation costs $O(\log n)$ amortized time [17]. Once $\overline{f^i}(x)$ is known, $r(u_i, D(G'))$ can be computed in $O(\log n)$ time for any u_i. Therefore,

Lemma 4.7. *After a preprocessing step requiring $O(n \log^2 n)$ time, the block of G'_{i_1} containing α_1'', say $B_{u_i^*}$, can be found in $O(\log^2 n)$ time. The storage space requirement is $O(n \log n)$.*

Locating α_1'' in the block $B_{u_i^*}$.

We will now prune away most of the edges of $B_{u_i^*}$ by using the information of optimal local centers on edges. Note that u_{i^*} is the closest hinge vertex of $B_{u_i^*}$ to v_{i_1}. We partition the edges of $B_{u_i^*}$ into two groups if $B_{u_i^*}$ is a cycle block: $\pi_{ccw} = [u_{i^*}, \ldots, u'_{i^*}]$ and $\pi_{cw} = [u_{i^*}, \ldots, u'_{i^*}]$, where u'_{i^*} is the match-point of u_{i^*} and the edges of π_{ccw} and π_{cw} are traversed in counterclockwise and clockwise orientations respectively. The edges of π_{ccw} and π_{cw} are, therefore, each ordered in increasing order from u_{i^*}. Note that when $B_{u_i^*}$ is a graft, only the edges on $P(u_i^*, u_{i+1}^*)$ need to be considered, and the procedure for this case is similar to the one described in the following. Thus, we assume that $B_{u_i^*}$ is a cycle block in the following.

We just consider π_{ccw} chain only. The process is similar for the other chain. The following lemma eliminates some edges of $B_{u_i^*}$.

Lemma 4.8. *If the optimal local center q of G'_{i_1} on an edge $e \in \pi_{ccw}$ has a larger service cost to G'_{i_1} than a point p on another edge of π_{ccw} closer to v_{i_1}, then α_1'' cannot lie on e.*

Proof. It is clear that $r(x, D(G_R^1)) = \max\{r(x, D(G')), r(x, D(G'_{i_1}))\}$ for any x in G'_{i_1}. Since the local minimum service cost to G'_{i_1} on e is greater than $r(p, D(G'_{i_1}))$ and p is closer to v_{i_1} than any point in e, the service cost $r(p, G_R^1)$ is always less than the service cost of any point on e. □

As a consequence of the above lemma we can order the edges of π_{ccw} in increasing distances from u_i^* and with decreasing optimal local center service costs. Similar ordering of the edges is performed on π_{cw}. These orderings are possible without the knowledge of G', and therefore are done once. The rest of the edges of B_i^* are labeled and will not be considered further. We only need to consider the unlabeled edges of $B_{u_i^*}$ to find α_1''. The following lemma allows us to prune the unlabeled edges of $B_{u_i^*}$ further.

Lemma 4.9. *Consider any unlabeled edge $e = \overline{uv}$ in π_{ccw} (u is closer to u_{i^*} than v) and its optimal local center q to G'_{i_1}. If the service cost of G_R^1 from q is determined by some vertex in G', then all the unlabeled edges in $\pi_{ccw}[v, \ldots, u'_{i^*}]$ cannot contain α_1''. Otherwise ($r(q, D(G')) < r(q, D(G'_{i_1}))$), all the unlabeled edges of $\pi_{ccw}[u_{i^*}, \ldots, u]$ cannot contain α_1''.*

The reason that we cannot directly use Lemma 4.9 on all the edges in π_{ccw} is that if $e = \overline{uv}$ is a labeled edge then it is possible to have the case where $r(q, G') < r(q, G'_{i_1})$ (q is the local center of e to G'_{i_1}) and α_1'' lies in $\pi_{ccw}[u_{i^*}, \ldots, u]$.

Therefore we can apply the binary-search technique to the unlabeled edges in π_{ccw} until one unlabeled edge is left, say e_{ccw}. We can similarly determine e_{cw} by performing a binary search on π_{cw}. Since the service cost of any point in G_R^1 to G_R^1 can be computed in $O(\log^2 n)$ time (Lemma A.1.2),

Lemma 4.10. *The number of candidate edges for α_1'' to lie on can be narrowed down to at most two in $O(\log^3 n)$ time.*

The remaining step of locating α_1'' on e_{ccw} and e_{cw} is very similar to computing the optimal local center of G'_{i_1} on an edge.

The above results can now be summarized as follows. After an $O(n \log^2 n)$-time processing, either we already have an optimal split-edge set or know the block B^* that contains an optimal split-edge set R^*. If B^* is a graft then it takes

an additional $O(n \log^2 n)$ time to compute an optimal split-edge and the optimal service cost. Otherwise, B^* is a cycle block. It is easy to see that finding G'_{i_1} and G'_{i_2} and adding match-points in them can be done in linear time. Due to the unimodality property of split-edges on a simple path, we only need to compute the service costs for $O(|B^*|)$ pairs of split-edges. After an $O(n \log^3 n)$-time preprocessing (which includes building two-level tree decomposition data structures and computing local centers), the service cost for each pair of split-edges can be computed in $O(\log^3 n)$ time, as we shown in Section 4.2.1. Therefore, we can claim that

Theorem 4.11. *The weighted $A(G)/V(G)/2$ problem in a cactus network can be solved in $O(n \log^3 n)$ time complexity. The storage space complexity is $O(n \log n)$.*

5. The *p*-center problems

As mentioned earlier, Frederickson and Johnson [12] designed an $O(n \log n)$-time algorithm for the unweighted $V(G)/V(G)/p$ problem in a cactus. They showed that the *feasibility test* in an unweighted cactus can be performed in linear time (Lemma 13 in [12]). A service cost t for the demand points in G is said to be *feasible* if there exists a set F of facilities of size p such that $r(F, D(G)) \leq t$. Using this linear-time feasibility test and a succinct representation of the set of all the inter-vertex distances, the unweighted p-center problem $V(G)/V(G)/p$ in a cactus network is solvable in $O(n \log n)$ time [12].

5.1. Weighted $V(G)/V(G)/p$ problem

Actually, the feasibility test described in [12] can also be applied for the case when the demand points (clients) in $D(G) = V(G)$ are weighted. In this case, for a given service cost t, the demand points may now have different covering radii. We present below a simple transformation that transforms the feasibility test in the weighted model to a feasibility test in a unweighted model.

In the weighted model, we have a cactus where each demand point node v_i is associated with a nonnegative covering radius $r_i = t/w_i$. The problem is to find a subset of nodes F of minimum cardinality, such that for each node v_i, $r(F, v_i) \leq r_i$. Lemma 13 in [12] provides an $O(n)$ algorithm for the case where $r_i = R'$, for each i. We can convert the above weighted model to an equivalent unweighted model as follows. Each node v_i of G is augmented by a new edge, say $\overline{v_i v'_i}$, of length $R' - r_i$, where $R' = \max\{r_j : v_j \in V(G)\}$. Let G' be the augmented graph with $2n$ nodes. G' is clearly a cactus. We now associate a radius R' with each node v_i and v'_i. The feasibility test on G is equivalent to a feasibility test on G', and therefore can be done in linear time. Thus

Lemma 5.1. *The feasibility test in a weighted model of the cactus can be performed in $O(n)$ time.*

Frederickson and Johnson [12] gave a succinct representation of the inter-vertex distances of the vertices of a cactus. The representation allows one to implement an efficient binary search on the distances. Similarly, the set of all inter-vertex distances in a partial 2-tree [13] has a special structure that enables searching the set without explicitly generating the entire set in advance. Indeed, the set of inter-vertex distances can be implicitly represented by a set of $O(n \log n)$ sorted lists. Each sorted list is associated with weighted distances from a given weighted vertex u to some subset V_u of the vertices of G whose shortest path distances to u pass through a separator vertex. These distances to the separator vertex are kept in sorted order. There are $O(\log n)$ such sorted lists for every node u. In this way the inter-vertex distances of any partial 2-tree can be represented by a set of $O(n \log n)$ sorted lists. This representation is very similar to the succinct representation of all inter-vertex distances in a tree proposed by Megiddo et al. [25]. Therefore, using the method proposed by Megiddo et al. [25], one can solve the discrete p-center problem in a weighted cactus graph for any p, in $O(n \log^2 n)$ time.

Theorem 5.2. *The weighted $V(G)/V(G)/p$ problem in a cactus graph can be solved in $O(n \log^2 n)$ time. The storage space requirement is $O(n \log n)$.*

5.2. Weighted $A(G)/V(G)/p$ and (unweighted) $V(G)/A(G)/p$ problems

From the fact that Lemma 5.1 is applicable also for the test corresponding to the weighted $A(G)/V(G)/p$ and the unweighted $V(G)/A(G)/p$ models, we can obtain an $O(n^2)$ algorithm for these problems. Since the numeric

operations of the feasibility test are additions/subtractions and comparisons, we can directly apply the generic parametric algorithm of Megiddo [21] and get the $O(n^2)$ time algorithm. Therefore

Theorem 5.3. *The weighted $A(G)/V(G)/p$ and the unweighted $V(G)/A(G)/p$ problems in a cactus network can be solved in $O(n^2)$ time.*

5.3. Unweighted $A(G)/A(G)/p$ problem

A candidate set containing the optimal solution value for the $A(G)/A(G)/p$ model for a general graph is characterized in the paper of Tamir [27]. In spite of the nice structure, this set is not of polynomial cardinality even for cactus networks. Nevertheless, in the discussion below we show that $A(G)/A(G)/p$ problem is efficiently solvable.

The idea is again to use the feasibility test parametrically (Megiddo [21]). First, we note that for this model p can be significantly larger than n. Nevertheless, the allocation of the p centers to the edges can be properly bounded. Let $p(e)$ denote the number of centers established at optimality on an edge e of length $l(e)$. Therefore, $(l(e)/2r^*) - 1 \leq p(e) \leq \lceil l(e)/2r^* \rceil + 1$, where r^* is the optimal service cost. It is shown in [27] that $p - m \leq L/2r^* \leq p + m$, where m and L are the number of edges and the total length of the edges in G respectively. Therefore

$$\max\{0, l(e)(p-m)/L - 1\} \leq p(e) \leq \min\{p, l(e)(p+m)/L + 1\}.$$

Hence $p(e)$ can a priori be bounded in a range of length $O(n)$ for cactus networks. In particular, when applying the test parametrically we will need $O(\log n)$ tests per edge to find the exact value of $p(e)$. An $O(n \log n)$ test for a more general class is mentioned in [14,27]. Therefore,

Theorem 5.4. *The $A(G)/A(G)/p$ problem in a cactus network can be solved in $O(n^2 \log^2 n)$ time.*

We remark that when the data of the above p-center problems are integer or even rational, and "relatively small", (e.g. sub-exponential in n), better complexity bounds can be achieved by applying an efficient search for rational techniques [31].

For the weighted $A(G)/V(G)/p$ model the optimal objective value is of the form $w(u)w(v)L(u,v)/(w(u) + w(v))$, where $L(u,v)$ is the length of some simple path connecting u and v for some pair of nodes $u, v \in V(G)$ [19]. For the $A(G)/A(G)/p$ model, the optimal solution value is of the form M/q, where M is the sum of the edge lengths of an Eulerian tour of some subgraph of G, and q is an integer, $1 \leq q \leq 4p$. The respective value for the $V(G)/A(G)/p$ model is of the form M/q, where M is the sum of the edge lengths of an Eulerian tour of some subgraph of G, and q is integer, $1 \leq q \leq 4$ [27].

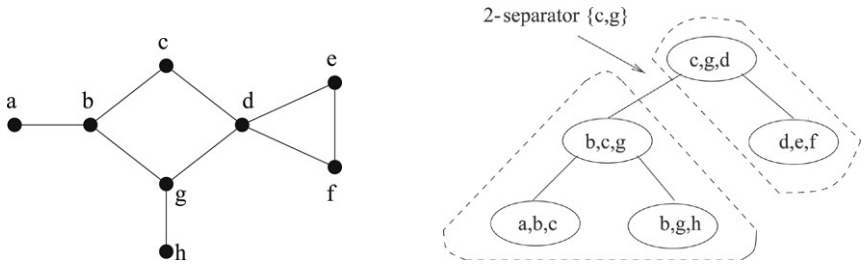
Assuming integer data, denote $W = \max_{v \in V(G)}\{w(v)\}$. Then, observing that $M \leq 2L$ and using the results in Zemel [31], we conclude that the weighted $A(G)/V(G)/p$, the $V(G)/A(G)/p$ and the $A(G)/A(G)/p$ problems can be solved in $O(n \log(n + L + W))$, $O(n \log(n + L))$ and $O(n \log n \log(n + L + p))$ times, respectively.

6. Conclusion and future work

In this paper we have studied the center problems in a tree-like network, a cactus, and proposed non-trivial algorithms to solve a variety of problems. The results are summarized in Table 2. We have proposed, for the first time, a sub-quadratic algorithm to solve the weighted continuous 1- and 2-center problems in a cactus network. Since, unlike trees, the service cost function on an edge is not convex in a cactus network, simple mechanism has been suggested that forces convexity on an edge in a cactus network. The convexity property allows one to compute the local minimum service cost on an edge in $O(\log^3 n)$ time that requires $O(n \log^2 n)$ time preprocessing. This is also true for other service cost functions such as the median cost and minimum cost.

The obnoxious center problem in a cactus network is to locate a facility in $A(G)$ such that the weighted minimum cost of the demand points is maximized. Since we can compute the local optimum minimum cost of the network on an edge in $O(\log^3 n)$ time,

Theorem 6.1. *The obnoxious center problem in a cactus network can be solved in $O(n \log^3 n)$ time.*



(a) a cactus graph G. (b) Tree decomposition $TD(G)$ of G.

Fig. A.1. A cactus graph and its corresponding tree decomposition with tree-width 2.

This improves the previous result of $O(cn)$, where c is the number of distinct vertex weights used in the graph [32]. In the worst case c is $O(n)$.

Many issues are still unresolved. For instance, it would be interesting to find out whether there exists an optimal linear-time algorithm for the weighted 1-center problem in a cactus graph.

We conjecture that the weighted $A(G)/V(G)/p$ problem and the unweighted $V(G)/A(G)/p$ and $A(G)/A(G)/p$ problems can be solved in subquadratic time by designing a polylog parallel algorithm for the feasibility test, and using the results in Megiddo [23]. For example, we suspect that the $O(\log^3 n)$ parallel time algorithm of Wang [30], for the test on trees, can be extended to cactus networks. If, indeed, there is an $O(\log^k n)$ parallel algorithm for cactus networks (with $O(n)$ processors), Megiddo [23] implies an $O(n \log^{k+1} n)$ serial algorithm for the weighted $A(G)/V(G)/p$ problem on cactus networks.

To obtain the result in Theorem 5.4 we have used an existing $O(n \log n)$ feasibility test. We suspect that an $O(n)$ test for $A(G)/A(G)/p$ can be derived by properly modifying the test for $V(G)/V(G)/p$ in [12]. This will lead to the improved bound $O(n^2)$ for $A(G)/A(G)/p$.

The most challenging problem is to find efficient algorithms to solve the weighted p-center problems in partial k-trees of bounded treewidth.

Acknowledgements

Research of the second author was partially supported by MITACS and NSERC.

Appendix A. A brief description of the two-level tree decomposition

One of the most important properties of trees, which is useful in designing efficient algorithms, is the existence of a 1-separator between any two disjoint subtrees. Partial k-trees form a more general class of graphs for which a similar property is available. Such a property is represented by a tree decomposition with bounded treewidth k, which can be found in linear time for fixed k [2]. In this case, there exists a k-separator between two subgraphs represented by two disjoint subtrees of this tree decomposition.

Cactus graphs are partial 2-trees. There is an efficient linear-time algorithm to get a tree decomposition with treewidth 2 for a partial 2-tree [13]. Refer to Fig. A.1. Assume that a tree decomposition $TD(G)$ of G is known. Given a subgraph G' represented by a subtree of $TD(G)$, there is a 2-separator in G' between G' and a point outside G'. We preprocess the local information of G' so that the service cost of an arbitrary facility (center) point located outside of G' to cover all the demand points in G' can be quickly computed.

Refer to Fig. A.2. Given any point p outside of G', the service cost to cover $v \in V(G')$ is either $w(v)\cdot(d_{v,u_1}+d_{u_1,p})$ or $w(v)\cdot(d_{v,u_2}+d_{u_2,p})$, where $\{u_1, u_2\}$ is the 2-separator of G'. Now the question is, for a given p outside G', which of the paths to v should be used as the shortest path? Suppose $a = d_{p,u_1} - d_{p,u_2}$ and $b = d_{v,u_1} - d_{v,u_2}$. Clearly the shortest path from v to p will go through u_1 if $a + b$ is negative; otherwise, the shortest path from v to p will go through u_2. For each G' of $TD(G)$ we create two sorted lists of increasing shortest path distance difference of all the vertices in G' to the separator vertices u_1 and u_2: $\delta_1(v) = d_{v,u_1} - d_{v,u_2}$ and $\delta_2(v) = d_{v,u_2} - d_{v,u_1}$ for all $v \in G'$. These two lists δ_1 and δ_2 are associated with the separator vertices u_1 and u_2 respectively. We build two balanced binary search trees on sorted lists δ_1 and δ_2, (see Fig. A.2(b)). Each search tree node represents the set of vertices of G' whose

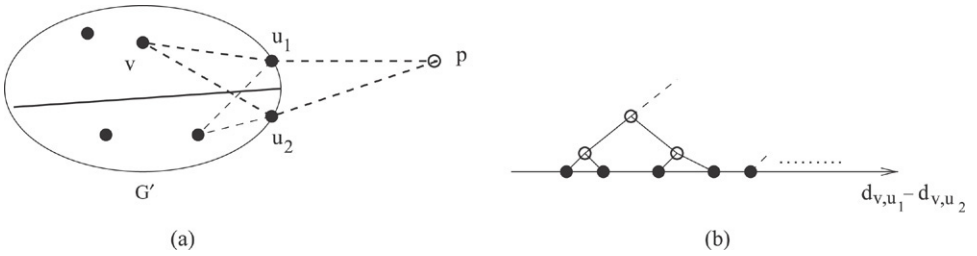


Fig. A.2. 2-separator between p and G'.

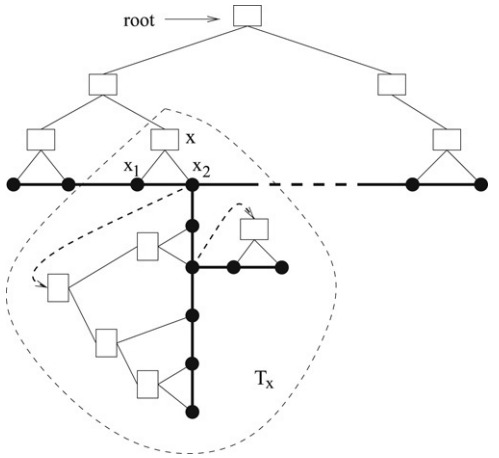


Fig. A.3. An example of spine tree decomposition.

shortest path differences are in its subtree. At each node of the balanced binary search tree corresponding to δ_1 (δ_2), we pre-compute the service cost function (weighted) of the vertices represented by the search tree node with the origin at u_1 (respectively u_2). The preprocessing step of computing all the service cost functions takes $O(|G'| \log |G'|)$. For a given point p outside G' that covers the vertices G' via $\{u_1, u_2\}$, there are $O(\log |G'|)$ sublists of vertices, determined by δ_1, whose shortest paths to p go through u_1, and there are $O(\log |G'|)$ sublists of vertices, determined by δ_2, whose shortest paths to p go through u_2. These lists can be identified in $O(\log |G'|)$ time. The maximum service cost of p to the vertices in a sublist can be computed in logarithmic time. Therefore, the total cost of computing the maximum service cost to the vertices in G' is $O(\log^2 |G'|)$. Note that the distance of p to u_1 and u_2 can be computed in constant time after almost linear time preprocessing [6].

Moreover, if we apply fractional cascading technique [8,9] on δ_1 and δ_2, the cost of computing the maximum service cost to G' for a given point outside G' can be improved into $O(\log |G'|)$, briefly described as follows. Consider the service cost function of a binary-search tree node as a list of intervals, where each interval is dominated by one vertex. By fractional cascading technique, the intervals in $O(\log |G'|)$ sublists for a given point p outside G' can be computed in $O(\log |G'|)$ time after $O(|G'| \log |G'|)$ preprocessing time. At each interval, the service cost from p to the sublist containing that interval is computable in constant time. In this way, the service cost to G' for a given point outside G' can be done in $O(\log |G'|)$ time.

Since the tree decomposition $TD(G)$ of G need not be balanced, we add another balanced tree structure over $TD(G)$ such that the height of the new tree $\overline{TD}(G)$ is logarithmic. We call the balanced tree structure of $TD(G)$ *a two-level tree decomposition of* G. There are several methods to achieve this, such as centroid tree decomposition, top-tree decomposition, and spine tree decomposition [1]. Here, we prefer the spine tree decomposition which rearranges $TD(G)$ to $\overline{TD}(G)$ because any algorithm developed using $TD(G)$ extends to $\overline{TD}(G)$ naturally. Please refer to [1] for more details. In Fig. A.3, the bold part is $TD(G)$. After the two-level tree decomposition of G, for each node in $TD(G)$, there are $O(\log n)$ rooted subtrees of $\overline{TD}(G)$ containing all the other nodes in $TD(G)$. Another important property of spine tree decomposition is that each rooted subtree of $\overline{TD}(G)$, say T_x rooted at x, communicates with the rest of the nodes in $TD(G)$ via at most two nodes x_1 and x_2 of T_x. Hence, the preprocessing step for each rooted

subtree (corresponding to G' in G) takes $O(|G'| \log |G'|)$ time (at most two 2-separators need to be considered). Thus, we get the following lemma:

Lemma A.1.1. *The complete two-level tree decomposition data structure of a partial 2-tree can be computed in $O(n \log^2 n)$ time requiring $O(n \log n)$ storage space.*

In particular, for each vertex in $V(G)$ there are $O(\log n)$ subgraphs, represented by rooted subtrees of $\overline{TD}(G)$, that contain the rest of the vertices in G. Moreover, for every point p in $A(G)$ (let $\overline{uv}$ be the edge containing p and let x be the node in $TD(G)$ containing both u and v) there are $O(\log n)$ rooted subtrees of $\overline{TD}(G)$ containing all the vertices in G.

Since distance queries in partial k-trees when k is fixed can be answered in constant time after almost linear-time preprocessing [6], the following lemma is easy to establish.

Lemma A.1.2. *The service cost of a point in a partial 2-tree can be answered in $O(\log^2 n)$ by the two-level tree decomposition data structure.*

It is not hard to see that this result can be extended to other service cost functions such as the median cost and minimum cost (obnoxious).

References

[1] R. Benkoczi, Cardinality constrained facility location problems in trees, Ph.D. Thesis, School of Computing Science, SFU, Canada, 2004.
[2] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[3] R.E. Burkard, H. Dollani, Center problems with pos/neg weights on trees, European J. Oper. Res. 145 (2003) 483–495.
[4] R.E. Burkard, H. Dollani, Y. Lin, G. Rote, The obnoxious center problem on a tree, SIAM J. Discrete Math. 14 (2001) 498–509.
[5] R.E. Burkard, J. Krarup, A linear algorithm for the pos/neg-weighted 1-median problem on cactus, Comput. 60 (1998) 498–509.
[6] S. Chaudhuri, C.D. Zaroliagis, Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms, Algorithmica 27 (3) (2000) 212–226.
[7] M.-L. Chen, R.L. Francis, T.J. Lowe, The 1-center problem: Exploiting block structure, Transport. Sci. 22 (1988) 259–269.
[8] B. Chazelle, L. Guibas, Fractional cascading: I. a data structuring technique, Algorithmica 1 (1986) 133–162.
[9] B. Chazelle, L. Guibas, Fractional cascading: II. applications, Algorithmica 1 (1986) 163–191.
[10] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, J. ACM 34 (1987) 200–208.
[11] G.N. Frederickson, Parametric search and locating supply centers in trees, in: Workshop on Algorithms and Data Structures, WADS, 1991, pp. 299–319.
[12] G.N. Frederickson, D.B. Johnson, Finding kth paths and p-centers by generating and searching good data structures, J. Algebra 4 (1983) 61–80.
[13] D. Granot, D. Skorin-Kapov, On some optimization problems on k-trees and partial k-trees, Discrete Appl. Math. 48 (1994) 129–145.
[14] Y. Gurevich, L. Stockmeyer, U. Vishkin, Solving NP-hard problems on graphs that are almost trees and an application to facility location problems, J. ACM 31 (3) (1984) 459–473.
[15] F. Harary, Graph Theory, Addison-Wesley, Reading, MA, 1969.
[16] R. Hassin, A. Tamir, Efficient algorithms for optimization and selection on series–parallel graphs, SIAM J. Alg. Discrete Math. 7 (3) (1986) 379–389.
[17] J. Hershberger, S. Suri, Off-line maintenance of planar configurations, J. Algebra 21 (1996) 453–475.
[18] M. Jeger, O. Kariv, Algorithms for finding p-centers on a weighted tree (for relatively small p), Networks 15 (1985) 381–389.
[19] O. Kariv, S.L. Hakimi, An algorithmic approach to network location problems, Part I. The p-centers, SIAM J. Appl. Math. 37 (1979) 513–538.
[20] Y.-F. Lan, Y.-L. Wang, H. Suzuki, A linear-time algorithm for solving the center problem on weighted cactus graphs, Inform. Process. Lett. 71 (1999) 205–212.
[21] N. Megiddo, Combinatorial optimization with rational objective functions, Math. Oper. Res. 4 (1979) 414–424.
[22] N. Megiddo, Linear-time algorithms for linear programming in R^3 and related problems, SIAM J. Comput. 12 (4) (1983) 759–776.
[23] N. Megiddo, Applying parallel computations algorithms in the design of serial algorithms, J. ACM 30 (1983) 852–865.
[24] N. Megiddo, A. Tamir, New results on the complexity of p-center problems, SIAM J. Comput. 12 (4) (1983) 751–758.
[25] N. Megiddo, A. Tamir, E. Zemel, R. Chandrasekaran, An $O(n \log^2 n)$ algorithm for the kth longest path in a tree with applications to location problems, SIAM J. Comput. 10 (2) (1981) 328–337.
[26] M.B. Rayco, R.L. Francis, A. Tamir, A p-center grid-positioning aggregatin procedure, Comput. Oper. Res. 26 (1999) 1113–1124.
[27] A. Tamir, On the solution value of the continuous p-center problem on a graph, Math. Oper. Res. 12 (1987) 345–363.
[28] A. Tamir, Improved complexity bounds for center location problems on networks by using dynamic data structures, SIAM J. Discrete Math. 1 (3) (1988) 377–396.
[29] A. Tamir, Obnoxious facility location on graphs, SIAM J. Discrete Math. 4 (1991) 550–567.
[30] B.-F. Wang, Finding r-dominating sets and p-centers on trees in parallel, IEEE Trans. Par. Distr. Syst. 15 (2004) 687–698.
[31] E. Zemel, On search over rationals, Oper. Res. Lett. 1 (1981) 34–38.
[32] B. Zmazek, J. Žerovnik, The obnoxious center problem on weighted cactus graphs, Discrete Appl. Math. 136 (2004) 377–386.